Software architecture analysis of usability

Folmer, Eelke

Chapter 10

Conclusions

In the introduction of this thesis a number of challenges concerning software architecture and software quality; usability in particular, were identified. One of the key problems with many of today's software products is that they do not meet their quality requirements very well. In addition, software engineers have come to the understanding that quality is not something that can be easily "added" to a software product in a late stage, since software quality is determined and restricted by architecture design. More specifically, organizations have spent a large amount of time and money on fixing architecture-related usability problems during late stage development. Therefore software engineers have come to the understanding that usability is not primarily determined by interface design but that it is also restricted and fulfilled by software architecture design.

10.1 Research Questions

In this section we will discuss how the research questions in Chapter 1 have been addressed by the papers presented in the previous chapters. In the introduction one overall research question was formulated. Before addressing the main research question, we will first discuss the research questions RQ-1 and RQ-2. For each of these research questions we first answer the four more detailed sub research questions before answering the research question itself.

10.1.1 Investigating the relationship between SA and usability

RQ-1.1: How does software architecture design restrict usability?

Our survey in Chapter 2 argues that the general notion in software engineering was that quality requirements are to an extent restricted by architecture design, but we did not know how this restriction precisely worked for usability. When investigating this relationship, we identified that certain usability improving design solutions such as undo and multichanneling (which are described as architecture sensitive usability patterns in chapters 3 and 4, and as bridging patterns in chapter 5) inhibited the *retrofit* problem. Adding such solutions during late stage development often proves to be problematic because their implementation is to some extent restricted by the software architecture. These solutions are what we call "architecture sensitive" (see the next research question for a definition of this concept). Implementing such a solution often requires large parts of source code to be rewritten which can be very expensive during late stage development. Because of business constraints such as time to market and costs, certain usability-improving mechanisms are prevented from being implemented during that stage, which could have otherwise increased the level of usability for that system. In these cases the software architecture restricts the level of usability.

During our case studies, however, we noticed that this restriction sometimes also works the other way around; architecture design sometimes leads to usability problems in the interface and the interaction design. For example, in the Webplatform case study (chapter 8) we identified that the layout of a page (users had to fill in a form) was determined by the XML definition of a specific object. This led to a usability problem, because when users had to insert data, the order in which particular fields had to be filled in turned out to be very confusing.

RQ-1.2: What does architecture sensitive mean?

We determined that architecture sensitivity has two aspects:

- **Retrofit problem**: Adding a certain design solution has a "structural" impact. These solutions are typically implemented as new architectural entities (such as components, layers, objects, etc) or an extension of existing architectural entities and relations and responsibilities between these entities. To add such a solution, some "transformation" of the software architecture is required. This impact can work on different levels in the architecture. In chapter 5 we give examples for four types of architectural transformations: architectural style, architectural pattern, design pattern, and transformation of quality requirements to functionality. The impact of implementing such a design solution can be either one of those transformations or a combination. In chapter 5 we give for each type of transformation an example architecture sensitive usability pattern that uses such a transformation.

- **Architecture support**: In chapter 5 we also argue that certain solutions (such as providing visual consistency, see chapter 4) do not necessarily require an extension or restructuring of the architecture. It is possible to implement these otherwise, for example by imposing a design rule on the system that states that all screens should be visually consistent (which is a solution that may work if you only have a few screens, but such a rule is often hard to enforce). However this is not the most optimal solution as adding new screens or changing existing screens requires a lot of effort. Sometimes because of other requirements, such as modifiability requirements, a more elegant and practical solution is chosen. Visual consistency may be easily facilitated by the use of a separation-of-data-from-presentation mechanism such as using XML/XSLT for creating templates (see chapter 5 for more details on the implementation issues). A template can be defined that is used by all screens, when the layout of a screen needs to be modified only the template should be changed. In this case the choice for a particular solution is also driven by the need to be able to modify screens (modifiability).

In general, "hard to retrofit" should be associated with "requires a lot of effort to implement during late stage" regardless of which transformation type is required. Architecture sensitivity of a specific pattern for a specific system can be a relative notion and depends on the context in which a pattern is being applied. The architectural impact of a pattern on a given system very much depends on what already has been implemented in the system. If some toolkit or framework is used that already supports a pattern (such as for example struts framework supporting model view controller), the impact may be low while the impact can be high if no support is there. In that case, that pattern for that system is no longer considered to be "architectural"

because with little effort it could be implemented. The main ideas behind this definition of architecture sensitive is that one of the goals of software architecture analysis should be to get rid of the "architectural" i.e. to an extent we should get rid of the irreversibility (Fowler, 2003) in software designs.

RQ-1.3: How can we describe these architecture sensitive design solutions?

The core contribution of our research consists of 15 architecture-sensitive usability-improving design solutions which we have captured in the form of patterns. These patterns have been identified from various industrial case studies and modern software applications as well as from existing (usability) pattern collections. An initial collection was presented in chapter 3 and this collection has been extended and refined in chapters 4 and 5. Patterns are an effective way of capturing and transferring knowledge. Although we have collected patterns for different types of usability problems such as interface design (multiple views) or interaction design (wizard) problems, the pattern format we defined in chapter 4 allows us to describe these patterns in a consistent format, increasing the readability of our collection. In order to use these patterns for architecture design and analysis, we decided to put these patterns in the Software-Architecture-Usability (SAU) framework. An initial outline of the SAU framework was already presented in chapter 2 and this framework has been further refined in chapters 3 and 4. In chapter 6 we also extend the main ideas behind the SAU framework to the qualities security and safety. In order to make the pattern collection useful we needed to answer two questions for each pattern:

1. How does this pattern improve usability? (SA analysis)

2. How can we use these patterns to design for usability? (SA design)

In order to answer these two questions we defined two layers in this framework:

- **Properties layer**: The usability properties embody the heuristics and design principles that researchers in the usability field consider to have a direct influence on system usability. The properties tell us how to design for usability and patterns are connected to these properties to be able to use these patterns for design.

- **Attribute layer**: A number of usability attributes have been selected from literature that appear to form the most common denominator of existing notions of usability. The attributes tell us how usability can be measured and properties are connected to attributes to express how they improve or impair these parts of usability.

Both properties and attributes result from our extensive survey of existing design practice and literature presented in chapter 2.

RQ-1.4: Can this design knowledge be used to *improve* current architecture design for usability?

The SAU framework captures essential design solutions so that these can be taken into account during architectural design and evaluation. Randomly applying patterns and properties from the SAU framework does not automatically lead to a usable system. Not all patterns and properties may improve usability for a specific system or are

relevant for a particular system. It is up to a usability engineer to decide whether a particular pattern or property is relevant for the system under analysis. The choice for a particular pattern or property may also be based upon cost and trade-offs between different usability attributes or between usability and other quality attributes such as security or performance.

To optimally use the design knowledge captured in the SAU framework we advice the use of our SALUTA method (see chapter 7) that guides the architect in determining the required and provided usability and in applying patterns from the framework.

Architecture design, which includes activities such as deciding on tradeoffs between qualities, is often a complex non formalized process, much relying on the experience of senior software engineers. As inevitably tradeoffs need to be made during design not only between attributes of a quality (inter-quality tradeoffs such as for usability, efficiency of use and learnability) but also between qualities (intra-quality tradeoffs: for example between usability and security) architecture design would benefit from knowing exactly for a particular design solution how this may affect other qualities. In chapter 6 we extend some of the ideas that led to the definition of the SAU framework, such as the concepts of architecture sensitive patterns, attributes and properties to other qualities.

Using the answers of RQ-1.1 - RQ 1.4 we can answer RQ-1.

RQ-1: What is the relationship between software architecture and usability?

The relationship between software architecture and usability is twofold:

- From the perspective of the software architecture in RQ-1.1 and RQ-2.2, we discovered that a software architecture may restrict the level of usability that can be achieved by permitting certain usability improving solutions to be added during late stage.

- From the perspective of usability, when looking at the system in isolation, usability is determined by the following factors:

    - Information architecture: how is information presented to the user? (This could even be further divided into pure interface issues and data presentation / organization) issues.

    - Interaction architecture: how is functionality presented to the user?

    - System quality attributes: such as efficiency and reliability.

Software architecture design does affect all these issues. For example, the quality attributes such as performance or reliability are to a considerable extent defined by the software architecture. Software architecture design may also have a major impact on the interaction and information architecture.

Putting these two perspectives together, we conclude that in our work we have specifically focused on the first aspect of this relationship. We identified a set of usability-improving design solutions that have architectural implications. Some of these solutions deal with usability problems in the information and interaction

architecture. So to an extent the second aspect is also covered. Our research currently does not address two parts of this complex relationship:

- Designing a usable system is more than ensuring a usable interface; a slow and buggy system architecture with a usable interface is not considered usable, on the other hand the most reliable and performing system architecture is not usable if the user can't figure out how to use the system. Usability is to an extent also determined by system qualities such as efficiency and reliability. In our work we only consider these qualities from a user perspective (e.g. efficiency of use and reliability of use). A solution such as undo helps with efficiency of use and reliability of use but we do not consider how the system is performing. The use of certain mechanism such as an XML/XSLT parser for facilitating multichanneling may have a serious impact on the performance of the final system, as such mechanisms may take a lot of memory. However this is not included in our analysis. The reason for this is that analyses of such qualities have their own unique problems. Researchers have already developed numerous techniques for analyzing these qualities. In order to design a high quality system, a whole suite of quality attribute specific assessment techniques should be used of which our technique for usability could be used for the assessment of usability.

- As mentioned before, sometimes architecture design may unknowingly lead to usability problems. For example, during one of our case studies we identified that a usability problem occurred, because the underlying data model determined how pages were presented. Assessing how an architecture may lead to these "anomalous" usability problems in these specific cases is quite difficult as these problems are only discovered during detailed design.

Summarizing, the relationship between software architecture and usability is partially expressed by our SAU framework (RQ-1.3). It provides developers with a repository of usability-improving design solutions that can improve current architectural design for usability (RQ-1.4).

The next set of research questions deal with turning software architecture design into a repeatable *process*.

10.1.2 Development of an SA assessment technique for usability

RQ-2.1: Which techniques can be used for architectural analysis of usability?

In chapter 2 an overview is provided of usability evaluation techniques that can be used during the different stages of development, unfortunately, no documented evidence exists of assessment techniques focusing on analyzing software architectures for usability.

Although it's possible to use for example a "generic" scenario-based assessment technique such as SAAM (Kazman et al, 1994), ATAM (Kazman et al, 1998) or QASAR (Bosch, 2000), a specialized technique is more tailored to a specific quality and will often lead to more accurate results. For example guidelines and criteria are given for creating specific scenarios

RQ-2.2: How should usability requirements be described for architecture assessment?

Before a software architecture can be assessed, the required usability of the system needs to be determined. In chapters 2 and 7, we argue that existing usability specification techniques and styles are poorly suited for architectural assessment, as these specifications are either rather abstract or performance based. Requirements specified as such are hard to evaluate for a software architecture. How should we evaluate a requirement specified as "the user must be able to learn this application in 20 minutes" when at this stage there is only a software architecture design to evaluate? Requirements should be expressed in such a way that we can analyze an architecture for its support of these.

In chapter 7 a specification technique is proposed based on scenario profiles. Scenario profiles describe the semantics of software quality attributes by means of a set of scenarios. The primary advantage of using scenarios is that scenarios represent the actual meaning of a requirement. For example a requirement specified as: "this part of the application should be easy to learn" does not specify for which users, tasks or contexts of use (usability depends on these variables) this requirement should hold. Our usage scenarios allow a more specific fine-grained specification of requirements which allows us to use them for architectural analysis. An additional reason for using scenarios is that we have had successful experiences with scenarios for architectural assessment of modifiability (Bengtsson, 2002).

Our usage scenarios are similar to use cases, a recognized form of task descriptions in HCI describing user-system interactions. In our technique we define usage scenarios with a similar purpose, namely to describe user-system interaction, but, as that does not express the required usability (it only describes interaction), we relate them to our usability attributes in the SAU framework presented in chapter 4. For certain usability attributes, such as efficiency and learnability, tradeoffs have to be made during design; as it is often impossible to design a system that has high scores on all attributes. A purpose of usability requirements is therefore to explicitly specify the necessary levels for each of these attributes. One should be aware that usage profile creation does not replace existing requirements engineering techniques. Rather it transforms (existing) usability requirements into something that can be used for architecture assessment.

In chapter 9 we reported some experiences with creating scenario profiles. Transforming requirements to a usage profile is sometimes difficult as requirements are either not specified at all, very weakly specified or change during development. As this process is an activity that takes place on the boundary of both SE and HCI disciplines cooperation with a usability engineer is often required. Overall, we had positive experiences with this technique and in our opinion scenario profiles can give an accurate description of the required usability suitable for architecture assessment, if usability requirements have sufficiently been specified.

RQ-2.3: How can we determine the architecture's support for a usage scenario?

In order to determine how a software architecture can support a particular usage profile, we analyze the software architecture using the SAU framework presented in chapter 4. Certain patterns and properties may support a particular usage scenario. For example, if undo has been implemented for a certain task then it may improve the usability for that scenario. We therefore analyze whether patterns and properties from the SAU framework are present in the available architecture designs and design documentation. The number of patterns and properties that support a particular usage

scenario are an indication for the architecture support for this scenario. For each pattern and property we use the relationships in the SAU framework to analyze how specific attributes of usability are improved. Using the undo example we can analyze that it may improve reliability and efficiency. It is up to the analyst to decide whether this support is sufficient. And our technique can assist in the decision making process by showing exactly how the software architecture may support a scenario.

The quality of the assessment very much depends on the amount of evidence for patterns and property support that can be extracted from the architecture. In chapter 9, we argue that because of the often non-explicit nature of architecture design, the analysis strongly depends on having access to both design documentation and software architects; as design decisions are often not documented the architect may fill in the missing information on the architecture and design decisions that were taken.

RQ-2-4: Can architecture assessment successfully be used for improving the usability of the final system?

Our cases studies (chapter 8,9,10) show that it is possible to use SALUTA to assess software architectures for their support of usability. We compared the results of the analysis with the results of final user testing results. Several user tests have been performed for each case study, these were conducted by our partners in the STATUS project. The results of these tests fit the results of our analysis: the software architecture supports the right level of usability. Some usability issues came up that where not predicted during our architectural assessment. However, these could not be traced back to insufficient support of the architecture for usability and could hence be fixed very easily.

In chapter 9 we identified several threats to the validity of our approach:

- Usability is often not an explicit design objective; SALUTA focuses on architectural assessment of usability. Any improvement in usability of the final system in the case studies performed should not be solely accounted to our method. More focus on usability during development in general is in our opinion the main cause for an increase in observed usability.

- Accuracy of usage profile: Deciding what users, tasks and contexts of use to include in the usage profile requires making tradeoffs between all sorts of factors. The representativeness of the usage profile for describing the required usability of the system is open to dispute. Questions whether we have accurately described the system's usage can only be answered by observing users when the system has been deployed. An additional complicating factor is the often weakly specified usability requirements which make it hard to create a representative usage profile.

- Case studies: Only three case studies were done so the conclusions presented here would be strengthened by the use of a greater number of case studies.

We are not sure whether our assessment can actually improve the usability of a system as so many other factors are involved. To validate SALUTA we should not only focus on measuring an increase in the usability of the resulting product but we should also measure the decrease in costs spent on usability during maintenance.

The main contribution of SALUTA is that it provides software architects with a method to understand and reason about the effect of design decisions on the quality of the final system, at a time when it is still cheap to change these decisions. One of the goals of architecture design and analysis is to get rid of the irreversibility (Fowler, 2003) in software designs. With SALUTA, architects may design a software that allows for more "usability tuning" (e.g. adding usability solutions with little effort) on the detailed design level. This may prevent part of the high costs incurred by adaptive (Swanson, 1976) maintenance activities once the system has been implemented. And if the costs for fixing usability issues are decreased, ensuring a usable system becomes a feasible option.

Using the answers of RQ-2.1 - RQ 2.4 we can answer RQ-2.

RQ-2: How can we assess a software architecture for its support of usability?

As there were no usability assessment techniques (RQ-2.1) focusing on the software architecture, we decided to develop our own technique. In chapter 7 we present a scenario based assessment technique SALUTA, which provides the analyst with a set of clearly defined steps for performing the analysis. Scenario profiles are used (RQ-2.2) for specifying the required usability. SALUTA uses the SAU framework (RQ-1) to identify the architecture support for usability (RQ-2.3). SALUTA can be used to make sure the software architecture supports usability which may save on maintenance costs (RQ-2.4).

10.1.3 Overall research question

Using the answers to RQ-1 and RQ-2 we can formulate an answer to the overall research question:

Given the fact that the level of usability is restricted and determined by the software architecture how can we design a software architecture that supports usability?

This main research question can now be answered as follows. The relevant knowledge needed for assessing and designing a software architecture for usability has been captured in the SAU framework in chapters 3, 4 and 5. Using the SALUTA assessment technique presented in chapters 7, 8 and 9 software architects can now iteratively assess and improve their software architectures.

10.2 Contributions

In the previous section about the research questions, we have already listed the contribution made by this thesis by answering the research questions. In this section we will summarize the contributions.

10.2.1 Investigating the relationship between SA and usability

- Until recently the relationship between software architecture and usability was poorly understood and described. Chapter 2 is a "call to arms" for investigating this relationship as it identifies that a large part of maintenance costs is spent on dealing with architecture related usability issues.

- In chapter 3 we present the first results of that research: a collection of architecture sensitive usability patterns and a basic outline of the SAU framework.

- Chapter 4 in detail describes the SAU framework, consisting of 15 patterns and 10 properties.

- In chapter 5 the work on architecture sensitive patterns is continued by presenting four bridging patterns. Bridging patterns also provide detailed implementation issues for interaction design patterns.

- Chapter 6 applies some of the concepts behind the definition of the SAU framework to other qualities. We illustrate this by presenting architecture sensitive patterns for security, safety and usability and show how inter- and intra- quality tradeoffs must be made between these qualities for implementing these patterns. In addition we present a "boundary" pattern which provides a solution to a quality problem (security) but also provides a solution that counters the negative effects on another quality (usability) that traditionally comes with implementing this pattern.

10.2.2 Development of an SA assessment technique for usability

- Architecture assessment of usability is necessary to be able to cost-effectively develop usable systems (as argued in chapter 2), as during this phase design decisions are made that are hard to change. In chapter 2, we also identify that there are no assessment techniques explicitly focusing on architecture assessment of usability.

- Chapter 7 addresses to this shortcoming by presenting a scenario-based architecture-level usability assessment technique called SALUTA. The steps of SALUTA are illustrated with a case study performed in the domain of web-based content management systems (Webplatform). To the best of our knowledge, SALUTA is the first architectural assessment technique that specifically focuses on usability.

- Chapter 8 presents two other case studies (eSuite, Compressor) performed with SALUTA at our industrial partners in the STATUS project.

- Chapter 9 presents a set of experiences and problems we encountered when conducting architecture analysis of usability. For each experience, a problem description, examples, causes, solutions and research issues are identified.

10.3 Future Work and Open Issues

Various issues discussed in this thesis can be subject of further research, as research often raises new research questions. Below, we will briefly discuss some of the areas that require further research:

- **Processes for software engineering and HCI are still not fully integrated**: The software architecture is seen as an intermediate product in the development process, but its potential with respect to quality assessment is not fully exploited. Software architects still often fail to associate usability with

software architecture design. When designing a system, software architects often already select technologies (read features) and often already develop a first version of the system before they decide to include the user in the loop. A software product is often seen as a set of features rather then a set of "user experiences". There are still many challenges open in changing these attitudes.

- **Maintenance costs:** Architectural assessment of usability as claimed in this thesis should save on maintenance costs. However, at the moment we lack figures that acknowledge this claim. In the organizations that participated in the case studies, these figures have not been recorded nor did there exist any historical data. To raise awareness and change attitudes (especially those of the decision makers), we should clearly define and measure the business advantages of architectural assessment of usability.

- **SAU framework**: Empirical validation is important when offering new techniques. The analysis technique for determining the provided usability of the system relies on the framework we developed. Initially, the SAU framework was based on discussions with our partners in the STATUS project and did not focus on any particular application domain. The list of patterns and properties that we had identified then was substantial but incomplete. As the goal of our research was mainly web based systems we refined the framework for this domain. Furthermore, our SAU framework initially contained usability patterns such as multitasking and shortcuts. For these patterns we could not find evidence that they were architecturally sensitive in this domain. Other patterns such as undo and cancel have different meanings in web based interaction. The list of architecture-sensitive usability patterns and properties we identified in the SAU framework is substantial but incomplete, it does not yet provide a complete comprehensive coverage of all potential architecture-sensitive usability issues for all domains. The case studies have allowed us to refine and extend the framework for the domain of web based enterprise systems, and allowed us to provide detailed architectural solutions for implementing these patterns and properties (based on "best" practices). But in order to improve design for usability for other domains we should also define frameworks for other domains.

- **Qualitative nature of SAU framework**: Relationships have been defined between the elements of the framework. Effectively an architect is interested in how much a particular pattern or property will improve a particular aspect of usability in order to determine whether requirements have been met. Being able to quantify these relationships would greatly enhance the use of our framework. In order to get quantitative data we need to substantiate these relationships and to provide models and assessment procedures for the precise way that the relationships operate. However, we doubt whether identifying this kind of (generic) quantitative information is possible.

- **Increase frame of reference**: SALUTA has been applied at three case studies. The different characteristics and different analysis goals of these cases have provided us with a small frame of reference for interpreting the analysis result. We have seen architectures with significant better and significantly weaker support for usability. This provided us with enough information to decide whether a particular architecture could still be improved. In order to expand our frame of reference more case studies need to be done. Certain

patterns such as multiple views were present in all architectures we examined, whereas other patterns such as user modes were only present in one system.

- **Tool support**: Creating and evaluating all scenarios by hand is time consuming, in order to automate this process tools should be developed which allow:
  - Specifying attribute values over multiple contexts, users, and tasks and that will automatically determine a prioritization of attribute values for a usage profile.
  - Quantify relationships by allowing architects to put weights on the patterns and properties that they consider to be important.
  - Determination of which patterns and properties support a scenario and come up with some quantitative indication for the support of that scenario.
- **Extend SAU framework to other qualities**: The SAU framework has proven to work for architecture assessment of usability. For other qualities such as security and safety we already identified architecture sensitive patterns such as SSO and Warning (see chapter 6) and we have reasons to believe that capturing this relevant design knowledge and identifying which tradeoffs must be made will be beneficial to software architecture design. The framework we outline and present in chapter 6 is far from complete but is a first step in formalizing and raising awareness of the complex relation between software quality and software architecture.